Data Science @ ISiM

A Taxonomy of Data Science

by **Hilary Mason and Chris Wiggins**

Both within the academy and within tech startups, we've been hearing some similar questions lately: Where can I find a good data scientist? What do I need to learn to become a data scientist? Or more succinctly: What *is* data science?

We've variously heard it said that data science requires some command-line fu for data procurement and preprocessing, or that one needs to know some **machine learning** or stats, or that one should know how to `look at data'. All of these are partially true, so we thought it would be useful to propose one possible taxonomy — we call it the Snice* taxonomy — of what a data scientist does, in roughly chronological order: Obtain, Scrub, Explore, Model, and iNterpret (or, if you like, OSEMN, which rhymes with **possum**).

Different data scientists have different levels of expertise with each of these 5 areas, but ideally a data scientist should be at home with them all.

We describe each one of these steps briefly below:

1. **Obtain: pointing and clicking does not scale.**

   Getting a list of numbers from a paper via PDF or from within your web browser via copy and paste rarely yields sufficient data to learn something `new' by exploratory or predictive analytics. Part of the skillset of a data scientist is knowing how to obtain a sufficient corpus of usable data, possibly from multiple sources, and possibly from sites which require specific query syntax. At a minimum, a data scientist should know how to do this from the command line, e.g., in a **UN*X environment**. Shell scripting does suffice for many tasks, but we recommend learning a programming or scripting language which can support automating the retrieval of data and add the ability to make calls

asynchronously and manage the resulting data. **Python** is a current favorite at time of writing (Fall 2010).

APIs are standard interfaces for accessing web applications, and one should be familiar with how to manipulate them (and even identify hidden, 'internal' APIs that may be available but not advertised). Rich actions on web sites often use APIs underneath. You have probably generated thousands of API calls already today without even knowing it! APIs are a two-way street: someone has to have *written* an API — a syntax — for you to interact with it. Typically one then writes a program which can execute commands to obtain these data in a way which respects this syntax. For example, let's say we wish to query the NYT archive of stories in bash. Here's a command-line trick for doing so to find stories about Justin Beiber (and the resulting **JSON**):

```
curl 'http://api.nytimes.com/svc/search/v1/article?query=justin%20beiber&api-key=f7b4a1749764ae
```

This Gist brought to you by GitHub. Justin_Bieber_API_Call view raw

```
{
offset: "0"
-results: [
-{
body: "Maybe we should just let the children run the country, at least until the recession is o
byline: "By CHARLES M. BLOW"
date: "20100814"
title: "OP-ED COLUMNIST; Justin Bieber for President"
url: "http://www.nytimes.com/2010/08/14/opinion/14blow.html"
}
]
-tokens: [
"justin"
"beiber"
]
total: 1
}
```

This Gist brought to you by GitHub. Justin_Bieber_API_Result.json view raw

Now let's look for stories with the word 'data' in the title, but in python:

```
import sys, os
import urllib, urllib2
```

```
import json

def main(query, api_key):
    h = urllib.urlopen("http://api.nytimes.com/svc/search/v1/article?%s" % (urllib.urlencode({'a
    return json.loads(h.read())

if __name__ == '__main__':
    print main("title:data","f7b4a1749764aec0364b215c354e3a0f:18:25759498")
```

This Gist brought to you by GitHub. Beiber_API_Call.py view raw

2. **Scrub: the world is a messy place**

   Whether provided by an experimentalist with missing data and inconsistent labels, or via a website with an awkward choice of data formatting, there will almost always be some amount of data cleaning (or scrubbing) necessary before analysis of these data is possible. As with Obtaining data, herein a little command line fu and simple scripting can be of great utility. Scrubbing data is the least sexy part of the analysis process, but often one that yields the greatest benefits. A simple analysis of clean data can be more productive than a complex analysis of noisy and irregular data.

   The most basic form of scrubbing data is just making sure that it's read cleanly, stripped of extraneous characters, and parsed into a usable format. Unfortunately, many data sets are complex and messy. Imagine that you decide to look at something as simple as the geographic distribution of twitter users by self-reported location in their profile. Easy, right? Even people living in the same place may use different text to represent it. Values for people who live in New York City contain "New York, NY", "NYC", "New York City", "Manhattan, NY", and even more fanciful things like "The Big Apple". This could be an entire blog post (and will!), but how do you disambiguate it? (**Example**)

   Sed, **awk**, **grep** are enough for most small tasks, and using either Perl or Python should be good enough for the rest. Additional skills which may come to play are familiarity with databases, including their syntax for representing data (e.g., JSON, above) and for querying databases.

3. **Explore: You can see a lot by looking**

   Visualizing, clustering, performing dimensionality reduction: these are all part of `looking at data.' These tasks are sometimes described as "exploratory" in that no hypothesis is being tested, no predictions are attempted. Wolfgang Pauli would call these techniques "not even wrong," though they are hugely useful for getting to know your data. Often such methods inspire predictive analysis methods used later. Tricks to know:

   - *more* or *less* (though less is more): Yes, that **more** and **less**. You can see a lot by looking at your

data. Zoom out if you need to, or use unix's **head** to view the first few lines, or **awk** or**cut** to view the first few fields or characters.

- Single-feature **histograms** visually render the range of single features and their distribution. Since histograms of real-valued data are contingent on choice of binning, we should remember that they an art project rather than a form of analytics in themselves.
- Similarly, simple feature-pair **scatter plots** can often reveal characteristics of the data that you miss when just looking at raw numbers.
- Dimensionality reduction (**MDS**, **SVD**, **PCA**, **PLS** etc): Hugely useful for rendering high-demensional data on the page. In most cases we are performing 'unsupervised' dimensionality reduction (as in PCA), in which we find two-dimensional shadows which capture as much variance of the data as possible. Occasionally, low-dimensional regression techniques can provide insight, for example in this **review article describing the Netflix Prize**which features a scatterplot of movies (Fig. 3) derived from a regression problem in which one wishes to predict users' movie ratings.
- Clustering: Unsupervised machine learning techniques for grouping observations; this can include grouping nodes of a graph into "modules" or "communities", or inferring latent variable assignments in a generative model with latent structure (e.g., **Gaussian mixture modeling**, or **K-means**, which can be derived via a limiting case of Gaussian mixture modeling).

4. **Models: always bad, sometimes ugly**

Whether in the natural sciences, in engineering, or in data-rich startups, often the 'best' model is the most predictive model. E.g., is it `better' to fit one's data to a straight line or a fifth-order polynomial? Should one combine a weighted sum of 10 rules or 10,000? One way of framing such questions of model selection is to remember why we build models in the first place: to predict and to interpret. While the latter is difficult to quantify, the former can be framed not only quantitatively but empirically. That is, armed with a corpus of data, one can leave out a fraction of the data (the "validation" data or "test set"), learn/optimize a model using the remaining data (the "learning" data or "training set") by minimizing a **chosen loss function** (e.g., squared loss, hinge loss, or exponential loss), and evaluate this or another loss function on the validation data. Comparing the value of this loss function for models of differing complexity yields the model complexity which minimizes generalization error. The above process is sometimes called "empirical estimation of generalization error" but typically goes by its nickname: **"cross validation."** Validation does not necessarily mean the model is "right." As **Box** warned us, "all models are wrong, but some are useful". Here, we are choosing from among a set of allowed models (the `hypothesis space', e.g., the set of 3rd, 4th, and 5th order polynomials) which model complexity maximizes predictive power and is thus the least bad among our choices.

Above we mentioned that models are built to predict and to interpret. While the former can be assessed quantitatively (`more predictive' is `less bad') the latter is a matter of which is less ugly, and is in the mind of the beholder. Which brings us to…

5. **iNterpret: "The purpose of computing is insight, not numbers."**

   Consider the task of automated **digit** recognition. The value of an algorithm which can predict ’4′ and distinguish from ’5’ is assessed by its predictive power, not on theoretical elegance; the goal of machine learning for digit recognition is not to build a theory of ’3.’ However, in the natural sciences, the ability to predict complex phenomena is different from what most mean by ‘understanding’ or ‘interpreting.’

   The predictive power of a model lies in its ability to generalize in the quantitative sense: to make accurate quantitative predictions of data in new experiments. The interpretability of a model lies in its ability to generalize in the qualitative sense: to suggest to the modeler which would be the most interesting experiments to perform next.

   The world rarely hands us numbers; more often the world hands us clickstreams, text, graphs, or images. Interpretable modeling in data science begins with choosing a natural set of input features — e.g., choosing a representation of text in terms of a **bag-of-words**, rather than bag-of-letters; choosing a representation of a graph in terms of **subgraphs** rather than the spectrum of the **Laplacian**. In this step, domain expertise and intuition can be more important than technical or coding expertise. Next one chooses a hypothesis space, e.g., linear combinations of these features vs. exponentiated products of special functions or **lossy hashes** of these features’ values. Each of these might have advantages in terms of computational complexity vs interpretability. Finally one chooses a learning/optimization algorithm, sometimes including a “regularization” term (which penalizes model complexity but does not involve observed data). For example, interpretability can be aided by learning by **boosting** or with an **L1** penalty to yield sparse models; in this case, models which can be described in terms of a comprehensible number of nonzero weights of, ideally, individually-interpretable features. Rest assured that interpretability in data science is not merely a desideratum for the natural scientist.

   Startups building products without the perspective of multi-year research cycles are often both exploring the data and constructing systems on the data at the same time. Interpretable models offer the benefits of producing useful products while at the same time suggesting which directions are best to explore next.

   For example, at **bit.ly**, we recently completed a project to classify popular content by click patterns over time and topic. In most cases, topic identification was straightforward, e.g., identifying celebrity gossip (you can imagine those features!). One particular click pattern was difficult to interpret, however; with further exploration we realized that people were using bit.ly links on images embedded in a page in order to study their own real-time metrics. Each page load counted as a ‘click’ (the page content itself was irrelevant), and we discovered a novel use case ‘in the wild’ for our product.

Deep thoughts:

Data science is clearly a blend of the hackers' arts (primarily in steps "O" and "S" above); statistics and machine learning (primarily steps "E" and "M" above); and the expertise in mathematics and the domain of the data for the analysis to be interpretable (that is, one needs to understand the domain in which the data were generated, but also the mathematical operations performed during the "learning" and "optimization"). It requires creative decisions and open-mindedness in a scientific context.

Our next post addresses how one goes about learning these skills, that is: "what does a data science curriculum look like?"

* named after **Snice**, our favorite NYC café, where this blog post was hatched.

Thanks to **Mike Dewar** for comments on an earlier draft of this.